Hardware-friendly neural computation of symmetric boolean functions

Bernard GIRAU

Abstract

The theoretical and practical framework of *Field Programmable Neural Arrays* has been defined to reconcile simple hardware topologies with complex neural architectures: FPNAs lead to powerful neural models whose original data exchange scheme allows to use hardware-friendly neural topologies. This report addresses preliminary results in the study of the computation power of FPNAs. The computation of symmetric boolean functions (e.g. the n-dimensional parity problem) is taken as a textbook example. The FPNA concept allows successive topology simplifications of standard neural models for such functions, so that the number of weights is reduced with a factor up to n with respect to previous works.

Note: FPNAs have already been studied in LORIA reports [Gir99c, Gir99d]. Therefore the description of FPNAs in section 2 is useless for the readers of these previous reports. Indeed, this report is a detailed study of a result briefly mentioned in [Gir99c]. It slightly extends the results of the original work in [Gir99b].

1 Introduction

Various fast implementations of neural networks have been developed. A broad introduction to parallel neural computing may be found in [NS92]. The very fine-grain parallelism of neural networks uses many information exchanges, so that hardware implementations are more likely to fit neural computations. Nevertheless, digital hardware implementations of neural networks (on FPGAs[1], ASICs[2], neuro-computers, ..., see [Gir00, Mor95]) either handle simplified neural computations or simple neural architectures, or they have to limit themselves to few well-fitted neural architectures. An upstream work is preferable ([Gir00]): neural computation paradigms may be defined to counterbalance the main implementation problems, and the use of such paradigms naturally leads to neural models that are more tolerant of hardware constraints, without any additional limitation. Since the main implementation difficulties are linked to area-greedy operators and complex topologies, two kinds of *hardware-adapted neural computation paradigms* may be found. Several models, such as bit-stream neural networks ([Sal94, vDJST93]), allow to handle area-saving neural computations, whereas the work of [Gir99b] leads to complex neural processings based on simplified topologies.

The *Field Programmable Neural Arrays* of [Gir99b] are based on a FPGA-like approach: a set of resources whose interactions are freely configurable. These resources (links and neural operators) are defined so as to perform computations of standard neurons, but they behave in an *autonomous* way. As a consequence, numerous *virtual* links may be achieved thanks to the application of a

[1] Configurable hardware devices such as *Field Programmable Gate Arrays* offer a compromise between the hardware efficiency of ASICs and a software-like flexibility.

[2] *Application Specific Integrated Circuits*

multicast data exchange protocol to the resources of a sparse neural network. This new neural computation concept enables a simplified neural architecture to replace a virtual complex one.

The practical study of [Gir99b, Gir99d] shows that FPNAs lead to very efficient hardware implementations of neural networks. The theoretical study of FPNAs ([Gir99b, Gir99a]) includes the determination of their computation power. FPNAs appear as more powerful than standard multilayer models for the exact computation of discrete functions (i.e. FPNAs require less neural resources). Conversely, FPNAs are less powerful than standard multilayer models for the exact computation of continuous functions (i.e. the weights of the virtual links are constrained in a subspace). As for the problem of approximate computing, [Gir99b, Gir99d] show that the topological simplifications of FPNAs do not infer a significant loss of approximation capability.

This report shows how FPNAs allow simplified neural architectures to compute discrete functions. The case of symmetric boolean functions is studied. The textbook case of the parity problem is detailed. Section 2 shortly describes the FPNA computation paradigm. Section 3 shows how this paradigm applies to symmetric boolean functions. Subsection 3.1 recalls previous results. Subsection 3.2 shows how the FPNA concept may be used so as to get rid of any shortcut link. Subsection 3.3 finally describes how a FPNA with $\mathcal{O}(\sqrt{n})$ weights may replace the quasi-optimal shortcut perceptron of [SRK91] that has $\mathcal{O}(n\sqrt{n})$ weights.

2 FPNAs

Two kinds of autonomous neural resources appear in a FPNA: *neurons* that apply standard neural functions to a set of input values on one hand, and communication *links* that behave as independent affine operators on the other hand. In a standard neural model, each communication link is a connection between the output of a neuron and an input of another neuron. The number of inputs of each neuron is its fan-in in the connection graph. On the contrary, communication links and neurons become *autonomous* in a FPNA: their dependencies are freely programmable.

More precisely, the communication links connect the nodes of a directed graph, each node contains one neuron. The specificity of FPNAs is that relations between *any* of the local resources of each node may be freely set. A link may be connected or not to the local neuron *and to the other local links*. Therefore direct connections between affine links appear, so that the FPNA may compute numerous composite affine transforms. These compositions create numerous *virtual neural links*.

2.1 Formal definition of FPNAs

A FPNA is defined[3] by means of:

- a directed graph $(\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is a finite set of nodes, and $\mathcal{E}$ is a set of directed edges without loop: for each node n, the set of the direct predecessors (resp. successors) of n is defined by $Pred(n) = \{p \in \mathcal{N} \mid (p,n) \in \mathcal{E}\}$ (resp. $Succ(n) = \{s \in \mathcal{N} \mid (n,s) \in \mathcal{E}\}$), the set of the input nodes is $\mathcal{N}_i = \{n \in \mathcal{N} \mid Pred(n) = \emptyset\}$,
- a set of neurons $((\theta_n, i_n, f_n))_{n\in\mathcal{N}}$, where $\theta_n \in \mathbb{R}$, i_n is a function from $\mathbb{R}^2$ to $\mathbb{R}$, and f_n is a function from $\mathbb{R}$ to $\mathbb{R}$: for each node there is one neuron resource that sequentially handles any neuron computation[4],

[3] This report presents a simplified FPNA definition which is sufficient as long as hardware implementations are not taken into account.

[4] Any standard neuron computation may be performed by means of a loop that updates a variable with respect to the neuron inputs, and a final computation that maps this variable to the neuron output. θ_n stands for the initialization value (see [Gir99b]). The iteration function i_n stands for the updating function inside the loop. The neuron output is finally computed with f_n.

- a set of affine functions $(x \mapsto W_n(p)x + T_n(p))_{(p,n)\in\mathcal{E}}$: for each node there are as many communication links as this node has got predecessors, each communication link is associated with an affine operator,
- for each node n in $\mathcal{N} - \mathcal{N}_i$,
  - a positive integer a_n: number of iterations before a neuron applies its transfer function,
  - for each p in $Pred(n)$, a binary value $r_n(p)$: set to 1 iff the link (p, n) and the neuron in n are connected,
  - for each s in $Succ(n)$, a binary value $S_n(s)$: set to 1 iff the neuron in n and the link (n, s) are connected,
  - for each p in $Pred(n)$ and each s in $Succ(n)$, a binary value $R_n(p, s)$: set to 1 iff the links (p, n) and (n, s) are connected,
- for each input node n in $\mathcal{N}_i$,
  - a positive integer c_n: number of global inputs sent by this node,
  - for each s in $Succ(n)$, a binary value $S_n(s)$ (see above).

2.2 Computing in a FPNA

Several computation methods have been defined for the FPNAs. In any such method, all resources behave independently, and when a resource receives values, it applies its local operator(s), and sends the result to all neighboring resources to which it is locally connected (a neuron resource waits for a_n values before sending any result to its neighbors). Unlike standard neural computations, the FPNA paradigm allows a resource to be connected or not to a neighboring resource. Moreover, a communication link may handle several values, and it may directly send them to other links.

The following sequential computation illustrates the basic FPNA principles. This computation method handles a list of tasks $\mathcal{L}$ that are processed according to a FIFO scheduling. Each task $[(p, n), x]$ corresponds to a value x sent on a communication link (p, n).

Initialization:

- For each input node n in $\mathcal{N}_i$, c_n values $\left(x_n^{(i)}\right)_{i=1..c_n}$ are given (global inputs of the FPNA), and the corresponding tasks $[(n, s), x_n^{(i)}]$ are created for all s in $Succ(n)$ such that $S_n(s) = 1$. The order of creation corresponds to a lexicographical order on (n, i, s) (with respect to the order of $\mathcal{N}$).
- Each node n in $\mathcal{N} - \mathcal{N}_i$ has got local variables c_n and x_n, initially set as $c_n = 0$ and $x_n = \theta_n$.

Sequential processing: *(while $\mathcal{L}$ is not empty)*

Let $[(p, n), x]$ be the first element in $\mathcal{L}$.

1. *remove this element from $\mathcal{L}$*
2. *compute $x' = W_n(p)x + T_n(p)$*
3. *for all $s \in Succ(n)$ such that $(R_n(p))(s) = 1$, create $[(n, s), x']$ according to the order on s*
4. *if $r_n(p) = 1$ (the neuron in n is said to be receiving the value of task $[(p, n), x]$)*
   - *update c_n and x_n : $c_n = c_n + 1$, $x_n = i_n(x_n, x')$*

- *if $c_n = a_n$ (the local neuron computes its output)*
  (a) $y = f_n(x_n)$, $c_n = 0$, $x_n = \theta_n$
  (b) *for all $s \in Succ(n)$ such that $S_n(s) = 1$, create $[(n,s),y]$ according to the order on s*

3 Symmetric boolean functions by FPNAs

3.1 A previous quasi-optimal solution

The neural computation of symmetric boolean functions has been a widely discussed problem. The quasi-optimal results of [SRK91] answer a question that was posed as early as in [Kau61]. A boolean function $f : \{0,1\}^d \to \{0,1\}$ is said symmetric if $f(x_1,\ldots,x_d) = f(x_{\sigma(1)},\ldots,x_{\sigma(d)})$ for any permutation σ of $\{1,\ldots,d\}$. An example is the d-dimensional parity problem: it consists in classifying vectors of $\{0,1\}^d$ as *odd* or *even*, according to the number of non zero values among the d coordinates.

This problem may be solved by d-input multilayer perceptron (MLP) or shortcut perceptrons. A MLP consists of several ordered layers of sigmoidal neurons. Two *consecutive* layers are fully connected. A shortcut perceptron also consists of several ordered layers of sigmoidal neurons. But a neuron in a layer may receive the outputs of the neurons in *all* previous layers. A layer which is not the input layer nor the output layer is said *hidden*. A MLP is indeed a special case of shortcut perceptron, where the weights of the shortcut links are zero (links between non-consecutive layers).

The search for optimal two-hidden layer shortcut perceptrons in [SRK91] has led to solve the d-dimensional parity problem with only $\sqrt{d}(2 + o(1))$ neurons, thanks to an iterated use of a method introduced in [Min61]. The shortcut links and the second hidden layer are essential in this work, though there is no shortcut link towards the output neuron. This neural network uses $d(2\sqrt{d} + 1 + o(1))$ weights. Previous results used $\mathcal{O}(d)$ neurons and $\mathcal{O}(d^2)$ weights. The results of [SRK91] apply to any symmetric boolean function.

Figure 1 shows the topology of the optimal shortcut network of [SRK91] for the 15-dimensional parity problem. In such a neural network, the first hidden layer may contain $\left\lceil \sqrt{d} \right\rceil$ neurons such that the i-th neuron of this layer computes $y_{i,1} = \sigma\left(\sum_{j=1}^{d} x_j + \Theta_i\right)$, where $\sigma(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$. The second hidden layer contains at most $\left\lceil \frac{d+1}{\lfloor \sqrt{d} \rfloor} \right\rceil$ neurons such that the i-th neuron of this layer computes $y_{i,2} = \sigma\left(\sum_{j=1}^{\lfloor \sqrt{d} \rfloor} w_{i,j,2} y_{j,1} + (-1)^i \sum_{j=1}^{d} x_j\right)$. Finally the only output neuron computes $y = \sigma\left(\sum_{j=1}^{\sqrt{d}} w_{i,j,3} y_{j,2} + \Theta\right)$.

3.2 Removing the shortcut links

The construction in [SRK91] implies that for any (i,j), $(-1)^i$ and $w_{i,j,2}$ have opposite signs. Therefore all shortcut links (between the input and the second hidden layer) may be virtually replaced by some direct connections between incoming and outgoing links in the first hidden layer of a FPNA. This FPNA has got $d\sqrt{d}(1+o(1))$ weights, instead of $d\sqrt{d}(2+o(1))$ weights in [SRK91].

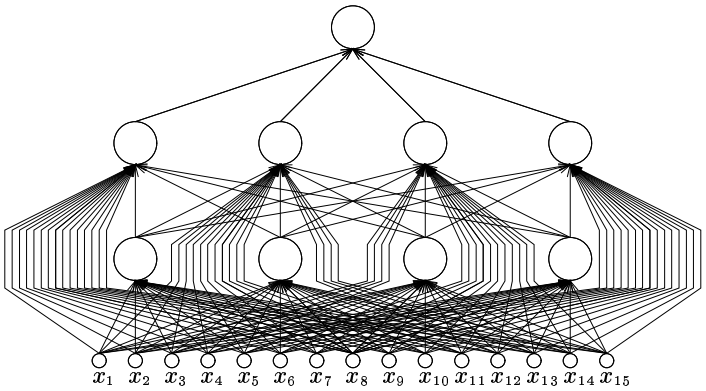

Figure 1: 2-hidden layer shortcut perceptron for the parity problem ($d = 15$)

More precisely, the architecture of the FPNA is the same as the shortcut perceptron of [SRK91], *without all shortcut links*. The weights of the links between both hidden layers are as in [SRK91]. Each neuron is fully connected to all incoming and outgoing links ($\forall\,(p,n)\; r_n(p) = 1$ and $\forall\,(n,s)\; S_n(s) = 1$). If n is the i-th node of the first hidden layer, and if s is the i-th node of the second hidden layer, then for any $p \in Pred(n)$, there is a direct connection between (p,n) and (n,s) (i.e. $(R_n(p))(s) = 1$). If n is the i-th node of the first hidden layer, then for any $p \in Pred(n)$, $W_n(p) = -\frac{1}{w_{i,i,2}}$ and $T_n(p) = 0$. If n is the i-th node of the first hidden layer, then $\theta_n = -\frac{\Theta_i}{w_{i,i,2}}$. Figure 2 sketches the architecture of such a FPNA for a 15-dimensional symmetric boolean function.

3.3 Towards a simplified 2D architecture

The topological simplifications allow to get rid of the shortcut links in the neural networks of [SRK91]. Yet the FPNA of figure 2 still does not have a hardware-friendly architecture (too many links and too large fan-ins). Designing a hardware-friendly FPNA for symmetric boolean functions involves a more drastic simplification of the architecture. The full connection scheme between consecutive layers may be virtually replaced by the use of sparse inter-layer and intra-layer links, thanks to the FPNA computation paradigm. The implementation of the required links fits a 2D device.

The construction of a FPNA in section 3.2 does not depend on the symmetric boolean function that is dealt with. On the contrary, the determination of the weights in a hardware-friendly FPNA for symmetric boolean functions takes advantage of function-dependent weight similarities in [SRK91]. Such FPNA constructions and weight determinations have been successfully performed for various symmetric boolean functions with different input dimensions, so that it appears that for any d and for any symmetric boolean function f, a FPNA with the same number of neurons as in [SRK91], but with only $\mathcal{O}(\sqrt{d})$ weights computes f *exactly* as in [SRK91]. Nevertheless, this assertion has not yet been formally proved. The parity problem may be taken as an example: in [SRK91], the weight of the link between the i-th neuron of the first hidden layer and the j-th neuron of the second hidden layer only depends on $(-1)^j$ when $i \neq 1$. The construction of a 2D FPNA for this problem does not depend on the input dimension (see [Gir99b]).

- The number of nodes in each hidden layer is the same as the number of neurons in [SRK91]. The number of input nodes is the number of nodes in the first hidden layer. Each input node

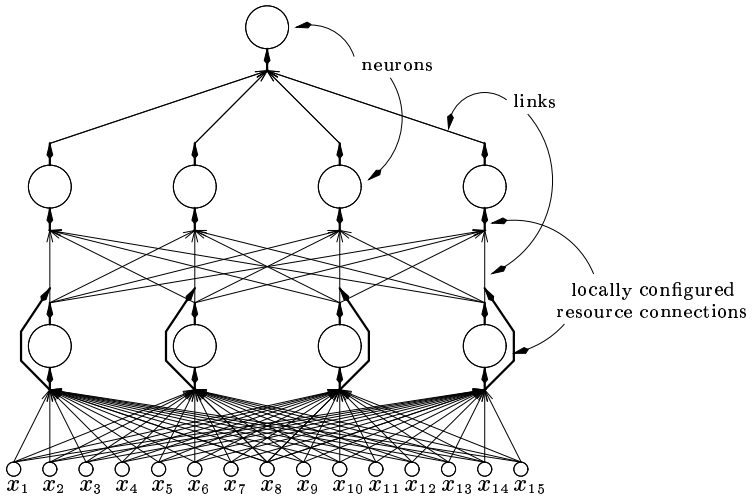

Figure 2: 2-hidden layer FPNA for the parity problem ($d = 15$)

(possibly except the last one) sends $\left\lceil \sqrt{d} \right\rceil$ input values x_j.

- The inter-layer links are: for any i, one link from the i-th input node towards the i-th node of the first hidden layer, and one link from the i-th node of the first hidden layer towards the i-th node of the second hidden layer, and one link from the i-th node of the second hidden layer towards the output node. Moreover, for any $j > 1$, there is a link between the first node of the first hidden layer and the $(2j-1)$-th node of the second hidden layer.
- The intra-layer links are: in both hidden layers, for any i, one link from the i-th node towards the $(i+1)$-th node, and another one towards the $(i-1)$-th node.
- The $S_n(s)$, $r_n(p)$ and $(R_n(p))(s)$ parameters are set so as to ensure a virtual full connection scheme between consecutive layers. Moreover the $(R_n(p))(s)$ parameters are set so that any virtual shortcut link involves the first node of the first hidden layer. See [Gir99b] for more details and for the weight determination.

This FPNA (for any number d of inputs) is easy to map onto a 2D hardware topology, whereas the equivalent shortcut perceptron defined in [SRK91] rapidly becomes too complex to be directly implemented when d increases. Figure 3 shows the architecture of the FPNA for the 15-dimensional parity problem.

Moreover, the theoretical study of [Gir99b, Gir99d] shows that the above FPNAs satisfy several conditions that ensure a computation time proportional to the number of weights as in standard multilayer models (such FPNAs are feedforward, deterministic and synchronous). Therefore, the FPNA paradigm allows to minimize the computation time of a symmetric boolean function by a neural network: previous works led to a $\mathcal{O}(d\sqrt{d})$ computation time, whereas a $\mathcal{O}(\sqrt{d})$ computation time is achieved thanks to the topological simplifications of the FPNAs.

4 Conclusion

The FPNA framework is a neural computation paradigm that has been defined in order to fit direct digital hardware implementations. The theoretical study of this original neural computation

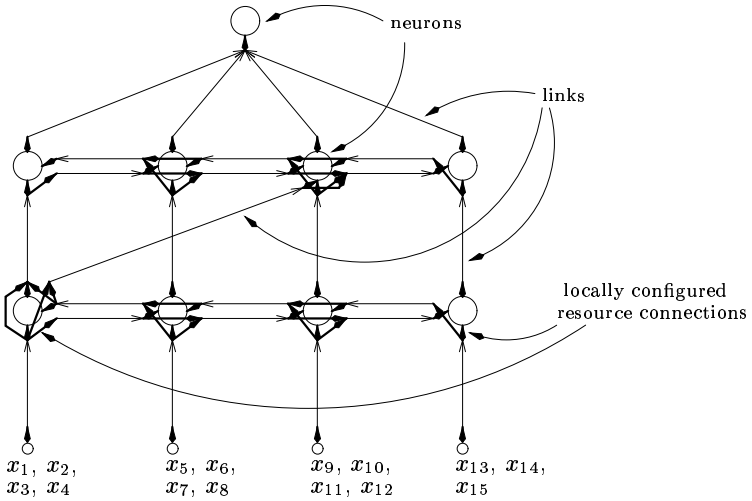

Figure 3: 2D FPNA for the parity problem ($d = 15$)

scheme includes many results whose interest lies far beyond the field of neural implementations. This report shows how FPNAs allow successive topological simplifications of the standard neural architectures that compute symmetric boolean functions. The shortcut links and then the full inter-layer connections are removed, and the FPNA computation protocol replaces them by virtual links based on numerous multicast composite connections. This work allows to compute d-dimensional symmetric boolean functions by neural models with $\mathcal{O}(\sqrt{d})$ weights, instead of $\mathcal{O}(d\sqrt{d})$ weights in the best previous works. In each case, the computation time is proportional to the number of weights. The results of [SRK91] are proved to be quasi-optimal in the number of neurons (so that the FPNAs defined above are also quasi-optimal). The question of the above FPNAs being quasi-optimal in the number of weights is now posed.

References

[Gir99a] B. Girau. Dependencies of composite connections in Field Programmable Neural Arrays. Research report NC-TR-99-047, NeuroCOLT, Royal Holloway, University of London, 1999.

[Gir99b] B. Girau. *Du parallélisme des modèles connexionnistes à leur implantation parallèle.* PhD thesis n° 99ENSL0116, ENS Lyon, 1999.

[Gir99c] B. Girau. FPNA, FPNN: from programmable fields to topologically simplified neural networks. Research report 99.R.019, LORIA INRIA-Lorraine, 1999.

[Gir99d] B. Girau. Synchronous FPNNs: neural models that fit reconfigurable hardware. Research report 99.R.143, LORIA INRIA-Lorraine, 1999.

[Gir00] B. Girau. Neural networks on FPGAs: a survey. In *Proc. Neural Computation*, 2000. To be published.

[Kau61] W. Kautz. The realization of symmetric switching functions with linear-input logical elements. *IRE Trans. Electron. Comput.*, EC-10, 1961.

[Min61] R. Minnick. Linear-input logic. *IEEE Trans. Electron. Comput.*, EC-10, 1961.

[Mor95] N. Morgan. Programmable neurocomputing systems. In *The Handbook of Brain Theory and Neural Networks*, pages 764–768. MIT Press, 1995.

[NS92] T. Nordström and B. Svensson. Using and designing massively parallel computers for artificial neural networks. *Journal of Parallel and Distributed Computing*, 14(3):260–285, 1992.

[Sal94] V. Salapura. Neural networks using bit-stream arithmetic: a space efficient implementation. In *Proc. IEEE Int. Conf. on Circuits and Systems*, 1994.

[SRK91] K. Siu, V. Roychowdhury, and T. Kailath. Depth-size tradeoffs for neural computation. *IEEE Trans. on Computers*, 40(12):1402–1412, 1991.

[vDJST93] M. van Daalen, P. Jeavons, and J. Shawe-Taylor. A stochastic neural architecture that exploits dynamically reconfigurable FPGAs. In *Proc. of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 202–211, 1993.